Contents

Circuit Breaking

7 minute read ✓ page test

This task shows you how to configure circuit breaking for connections, requests, and outlier detection.

Circuit breaking is an important pattern for creating resilient microservice applications. Circuit breaking allows you to write applications that limit the impact of failures, latency spikes, and other undesirable effects of network peculiarities.

In this task, you will configure circuit breaking rules and then test the configuration by intentionally “tripping” the circuit breaker.

Before you begin

- Setup Istio by following the instructions in the Installation guide.
- Start the httpbin sample.

  If you have enabled automatic sidecar injection, deploy the `httpbin` service:

  ```
  $ kubectl apply -f @samples/httpbin/httpbin.yaml@
  ```

  Otherwise, you have to manually inject the sidecar before deploying the `httpbin` application:

  ```
  $ kubectl apply -f <(istioctl kube-inject -f @samples/httpbin/httpbin.
  yaml@)
  ```

The `httpbin` application serves as the backend service for this task.

Configuring the circuit breaker

1. Create a destination rule to apply circuit breaking settings when calling the `httpbin` service:

   > ⚠ If you installed/configured Istio with mutual TLS authentication enabled, you must add a TLS traffic policy `mode: ISTIO_MUTUAL` to the `DestinationRule` before applying it. Otherwise

requests will generate 503 errors as described here.

```
$ kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: httpbin
spec:
  host: httpbin
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
    outlierDetection:
      consecutive5xxErrors: 1
      interval: 1s
      baseEjectionTime: 3m
      maxEjectionPercent: 100
EOF
```

2. Verify the destination rule was created correctly:

```
$ kubectl get destinationrule httpbin -o yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
...
spec:
  host: httpbin
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
      tcp:
        maxConnections: 1
    outlierDetection:
      baseEjectionTime: 3m
      consecutive5xxErrors: 1
      interval: 1s
      maxEjectionPercent: 100
```

Adding a client

Create a client to send traffic to the `httpbin` service. The client is a simple load-testing client called fortio. Fortio lets you control the number of connections, concurrency, and delays for outgoing HTTP calls. You will use this client to “trip” the circuit breaker policies you set in the `DestinationRule`.

1. Inject the client with the Istio sidecar proxy so network interactions are governed by Istio.

   If you have enabled automatic sidecar injection, deploy the `fortio` service:

   ```
   $ kubectl apply -f @samples/httpbin/sample-client/fortio-deploy.yaml@
   ```

Otherwise, you have to manually inject the sidecar before deploying the `fortio` application:

```
$ kubectl apply -f <(istioctl kube-inject -f @samples/httpbin/sample-c
lient/fortio-deploy.yaml@)
```

2. Log in to the client pod and use the fortio tool to call `httpbin`. Pass in `curl` to indicate that you just want to make one call:

```
$ export FORTIO_POD=$(kubectl get pods -l app=fortio -o 'jsonpath={.it
ems[0].metadata.name}')
$ kubectl exec "$FORTIO_POD" -c fortio -- /usr/bin/fortio curl -quiet
http://httpbin:8000/get
HTTP/1.1 200 OK
server: envoy
date: Tue, 25 Feb 2020 20:25:52 GMT
content-type: application/json
content-length: 586
```

```
access-control-allow-origin: *
access-control-allow-credentials: true
x-envoy-upstream-service-time: 36

{
  "args": {},
  "headers": {
    "Content-Length": "0",
    "Host": "httpbin:8000",
    "User-Agent": "fortio.org/fortio-1.3.1",
    "X-B3-Parentspanid": "8fc453fb1dec2c22",
    "X-B3-Sampled": "1",
    "X-B3-Spanid": "071d7f06bc94943c",
    "X-B3-Traceid": "86a929a0e76cda378fc453fb1dec2c22",
    "X-Forwarded-Client-Cert": "By=spiffe://cluster.local/ns/default/s
a/httpbin;Hash=68bbaedefe01ef4cb99e17358ff63e92d04a4ce831a35ab9a31d3c8
e06adb038;Subject=\"\";URI=spiffe://cluster.local/ns/default/sa/defaul
t"
  },
  "origin": "127.0.0.1",
  "url": "http://httpbin:8000/get"
}
```

You can see the request succeeded! Now, it’s time to break something.

Tripping the circuit breaker

In the `DestinationRule` settings, you specified `maxConnections: 1` and `http1MaxPendingRequests: 1`. These rules indicate that if you exceed more than one connection and request concurrently, you should see some failures when the `istio-proxy` opens the circuit for further requests and connections.

1. Call the service with two concurrent connections (`-c 2`) and

send 20 requests (`-n 20`):

```
$ kubectl exec "$FORTIO_POD" -c fortio -- /usr/bin/fortio load -c 2 -q
ps 0 -n 20 -loglevel Warning http://httpbin:8000/get
20:33:46 I logger.go:97> Log level is now 3 Warning (was 2 Info)
Fortio 1.3.1 running at 0 queries per second, 6->6 procs, for 20 calls
: http://httpbin:8000/get
Starting at max qps with 2 thread(s) [gomax 6] for exactly 20 calls (1
0 per thread + 0)
20:33:46 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:33:47 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:33:47 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
Ended after 59.8524ms : 20 calls. qps=334.16
Aggregated Function Time : count 20 avg 0.0056869 +/- 0.003869 min 0.0
00499 max 0.0144329 sum 0.113738
# range, mid point, percentile, count
>= 0.000499 <= 0.001 , 0.0007495 , 10.00, 2
> 0.001 <= 0.002 , 0.0015 , 15.00, 1
> 0.003 <= 0.004 , 0.0035 , 45.00, 6
> 0.004 <= 0.005 , 0.0045 , 55.00, 2
> 0.005 <= 0.006 , 0.0055 , 60.00, 1
> 0.006 <= 0.007 , 0.0065 , 70.00, 2
> 0.007 <= 0.008 , 0.0075 , 80.00, 2
> 0.008 <= 0.009 , 0.0085 , 85.00, 1
> 0.011 <= 0.012 , 0.0115 , 90.00, 1
> 0.012 <= 0.014 , 0.013 , 95.00, 1
> 0.014 <= 0.0144329 , 0.0142165 , 100.00, 1
# target 50% 0.0045
# target 75% 0.0075
# target 90% 0.012
# target 99% 0.0143463
# target 99.9% 0.0144242
Sockets used: 4 (for perfect keepalive, would be 2)
Code 200 : 17 (85.0 %)
Code 503 : 3 (15.0 %)
Response Header Sizes : count 20 avg 195.65 +/- 82.19 min 0 max 231 su
m 3913
Response Body/Total Sizes : count 20 avg 729.9 +/- 205.4 min 241 max 8
17 sum 14598
All done 20 calls (plus 0 warmup) 5.687 ms avg, 334.2 qps
```

It’s interesting to see that almost all requests made it through! The `istio-proxy` does allow for some leeway.

```
Code 200 : 17 (85.0 %)
Code 503 : 3 (15.0 %)
```

2. Bring the number of concurrent connections up to 3:

```
$ kubectl exec "$FORTIO_POD" -c fortio -- /usr/bin/fortio load -c 3 -q
ps 0 -n 30 -loglevel Warning http://httpbin:8000/get
20:32:30 I logger.go:97> Log level is now 3 Warning (was 2 Info)
Fortio 1.3.1 running at 0 queries per second, 6->6 procs, for 30 calls
: http://httpbin:8000/get
Starting at max qps with 3 thread(s) [gomax 6] for exactly 30 calls (1
0 per thread + 0)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
```

```
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
20:32:30 W http_client.go:679> Parsed non ok code 503 (HTTP/1.1 503)
Ended after 51.9946ms : 30 calls. qps=576.98
Aggregated Function Time : count 30 avg 0.0040001633 +/- 0.003447 min
0.0004298 max 0.015943 sum 0.1200049
# range, mid point, percentile, count
>= 0.0004298 <= 0.001 , 0.0007149 , 16.67, 5
> 0.001 <= 0.002 , 0.0015 , 36.67, 6
> 0.002 <= 0.003 , 0.0025 , 50.00, 4
> 0.003 <= 0.004 , 0.0035 , 60.00, 3
> 0.004 <= 0.005 , 0.0045 , 66.67, 2
> 0.005 <= 0.006 , 0.0055 , 76.67, 3
> 0.006 <= 0.007 , 0.0065 , 83.33, 2
> 0.007 <= 0.008 , 0.0075 , 86.67, 1
> 0.008 <= 0.009 , 0.0085 , 90.00, 1
> 0.009 <= 0.01 , 0.0095 , 96.67, 2
> 0.014 <= 0.015943 , 0.0149715 , 100.00, 1
# target 50% 0.003
# target 75% 0.00583333
# target 90% 0.009
# target 99% 0.0153601
# target 99.9% 0.0158847
Sockets used: 20 (for perfect keepalive, would be 3)
Code 200 : 11 (36.7 %)
Code 503 : 19 (63.3 %)
Response Header Sizes : count 30 avg 84.366667 +/- 110.9 min 0 max 231
 sum 2531
Response Body/Total Sizes : count 30 avg 451.86667 +/- 277.1 min 241 m
ax 817 sum 13556
All done 30 calls (plus 0 warmup) 4.000 ms avg, 577.0 qps
```

Now you start to see the expected circuit breaking behavior. Only 36.7% of the requests succeeded and the rest were trapped by circuit breaking:

```
Code 200 : 11 (36.7 %)
Code 503 : 19 (63.3 %)
```

3. Query the `istio-proxy` stats to see more:

```
$ kubectl exec "$FORTIO_POD" -c istio-proxy -- pilot-agent request GET
 stats | grep httpbin | grep pending
cluster.outbound|8000||httpbin.default.svc.cluster.local.circuit_break
ers.default.remaining_pending: 1
cluster.outbound|8000||httpbin.default.svc.cluster.local.circuit_break
ers.default.rq_pending_open: 0
cluster.outbound|8000||httpbin.default.svc.cluster.local.circuit_break
ers.high.rq_pending_open: 0
cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream_rq_p
ending_active: 0
cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream_rq_p
ending_failure_eject: 0
cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream_rq_p
ending_overflow: 21
cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream_rq_p
ending_total: 29
```

You can see `21` for the `upstream_rq_pending_overflow` value which means `21` calls so far have been flagged for circuit breaking.

Cleaning up

1. Remove the rules:

   ```
   $ kubectl delete destinationrule httpbin
   ```

2. Shutdown the `httpbin` service and client:

   ```
   $ kubectl delete deploy httpbin fortio-deploy
   $ kubectl delete svc httpbin fortio
   ```